Métodos Numéricos

11 de enero de 2026

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo práctico 1

Primer cuatrimestre 2024

Grupo 19

Integrante	LU	Correo electrónico
Rios, Brian Ivan	917/19	ivan.rios2010@gmail.com
Saccomano, Ignacio Manuel	222/22	nachosacco1@gmail.com
Maldonado, Juan Bautista	164/22	jbmaldonado2003@gmail.com

Resumen

Se modeló el estudio de la difusión de partículas como la resolución se sistemas de ecuaciones lineales con una matriz asociada tridiagonal. Al principio ahondamos sobre los métodos de resolución estándar (eliminación gaussiana con y sin pivoteo) y cómo manejar el error numérico. A partir de estos conceptos construimos una solución específica para este tipo de matrices resultando en una alternativa que tiene una complejidad lineal.

Palabras clave: Eliminación gaussiana, error, matrices tridiagonales, difusión.

Índice

1. Introducción **2**

2. Desarrollo **3**
2.1. Eliminación Gaussiana 3
2.2. EG con pivoteo parcial 5
2.3. Solución para el Caso Tridiagonal 6
2.4. Verificación de la implementación 8
2.5. Tiempos de cómputo 9
2.6. Simulación de difusión 11

3. Conclusiones **13**

4. Bibliografía **14**

5. Apéndice **15**

1. Introducción

En la naturaleza es común encontrarse con fenómenos donde una muestra se dispersa por un medio de manera estocástica, es decir, aleatoria. Existen diversos ejemplos, uno de ellos es la máquina de Galton [1]. Este problema interpela a muchas áreas del conocimiento como la física o las finanzas. Es por ello que es de particular interés poder modelar estos problemas de una manera tal que se pueda resolver fácilmente por una computadora en términos de complejidad computacional. Para ello es fundamental modelar matemáticamente qué es lo que sucede en un proceso de difusión.

Primeramente notar que la operación discreta del operador laplaciano en el caso unidimensional está dado por la siguiente ecuación que puede traducirse en una matriz tridiagonal [3]. Sea u un vector de dimensión n, donde n es la cantidad de posiciones a las que pueden llegar las partículas. Si se tienen n posiciones, se puede representar la densidad de partículas en la posición i con la componente u_i:

$$u_{i-1} - 2u_i + u_{i+1} = d_i \tag{1}$$

Al ser una densidad, se cumple que $0 \leq u_i \leq 1$ para todo $0 \leq i \leq n$. Para expresar la ecuación de difusión se tiene la siguiente ecuación, donde α describe el grado de difusión. [2]:

$$u_i^{(k)} - u_i^{(k-1)} = \alpha(u_{i-1}^{(k-1)} - 2u_i^{(k-1)} + u_{i+1}^{(k-1)}) \tag{2}$$

Otra formulación posible es la que considera que el incremento depende del laplaciano aplicado en la magnitud en el paso k, en vez de $k-1$:

$$u_i^{(k)} - u_i^{(k-1)} = \alpha(u_{i-1}^{(k)} - 2u_i^{(k)} + u_{i+1}^{(k)}) \tag{3}$$

Teniendo en cuenta las ecuaciones 2 y 3 se puede construir un sistema de ecuaciones con la matriz asociada al operador laplaciano adaptada para modelar el problema de difusión para ambos casos como se expondrá más adelante.

A continuación explicaremos cuál es el estado del arte para resolver estos sistemas, ahondando en los mecanismos que utilizan para construir las soluciones y cómo tratan el error numérico intrínseco a la aritmética finita. A partir de estos algoritmos se realizarán modificaciones que aprovechan la forma particular de las matrices asociadas al problema de difusión, dando lugar a una alternativa más eficiente e igual de precisa. Esto permitirá estudiar el fenómeno de la evolución de densidad de partículas requiriendo menos tiempo de cómputo y recursos, facilitando las simulaciones que este problema requiere.

2. Desarrollo

2.1. Eliminación Gaussiana

Para la resolución de sistemas de ecuaciones lineales el método por excelencia es la Eliminación Gaussiana (EG) que consta de dos partes.

La primera etapa es conocida como **forward elimination**, y consiste en la construcción de un sistema equivalente (i.e: un sistema de ecuaciones distinto que comparte las mismas soluciones) con la particularidad de que la matriz asociada resultante es triangular superior. Para la triangulación de la matriz se resta a cada fila un múltiplo adecuado de la anterior que deja a la columna correspondiente en 0. Expresado formalmente, al final de la iteración i se cumple que $m_{ij} = 0$ $(\forall i < j \leq n)$ donde m_{ij} es el coeficiente en la fila i y columna j de la matriz.

$$F_j^{(k)} = F_j^{(k-1)} - m_{ji}/m_{ii} F_i^{(k-1)} \qquad i < j \leq n \tag{4}$$

Donde k denota la fila resultante luego de haber ejecutado k iteraciones. Un ejemplo:

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ a_{21}^{(0)} & a_{22}^{(0)} & \cdots & a_{2n}^{(0)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(0)} & a_{n2}^{(0)} & \cdots & a_{nn}^{(0)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_n^{(0)} \end{bmatrix}$$

$$\vdots$$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix}$$

$$\vdots$$

$$\begin{bmatrix} a_{11}^{(k)} & a_{12}^{(k)} & \cdots & a_{1n}^{(k)} \\ 0 & a_{22}^{(k)} & \cdots & a_{2n}^{(k)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn}^{(k)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(k)} \\ b_2^{(k)} \\ \vdots \\ b_n^{(k)} \end{bmatrix}$$

El algoritmo entonces consiste en ejecutar la ecuación (7) para todas las filas desde la 2 a la n, ya que así se anulan todos los coeficientes de la columna i a partir de la fila $i + 1$ $(\forall i / 1 \leq i \leq n)$. Luego en la etapa de **backward substitution** se resuelven las incógnitas, empezando por la última y luego despejando las anteriores en función de las que les suceden.

Algorithm 1 Eliminación Gaussiana sin pivoteo

```
1: procedure FORWARD ELIMINATION(A, b)
2:     n ← dimensión de A
3:     for i ← 1 hasta n − 1 do
4:         for j ← i + 1 hasta n do
5:             if A_ii^(k−1) == 0 then
6:                 if col_i(A) == 0 then
7:                     break                ▷ La columna ya está anulada, no es necesario usar pivote
8:                 else
9:                     FIN                  ▷ No se puede triangular la matriz
10:                end if
11:            else
12:                m ← A_ij^(k−1) / A_ii^(k−1)
13:            end if
14:            for k ← j + 1 to n do
15:                A_jk^(k) ← A_jk^(k−1) − m A_kj^(k−1)
16:            end for
17:            b_j ← b_j − m b_i
18:        end for
19:    end for
20: end procedure
```

Algorithm 2 Eliminación Gaussiana

```
1: procedure BACKWARD SUBSTITUTION(A, b, x)
2:     n ← dimensión de A
3:     for i ← n hasta 1 do
4:         x_i ← (b_i − Σ_{j=i+1}^{n−1} A_ij x_j) / A_ii
5:     end for
6: end procedure
```

El primer ciclo, como fue mencionado anteriormente, cumple el invariante de que al terminar la i-ésima iteración la columna i tiene ceros debajo de la diagonal. El segundo ciclo consiste en encontrar el coeficiente adecuado para multiplicar por la fila i de modo tal que al restar las filas i y j quede un cero en la posición i. El tercer y último ciclo, entonces, se ocupa de realizar la resta de la ecuación anteriormente descrita [(7)]. Luego al realizar backward substitution se computan las soluciones.

Este algoritmo, sin embargo, puede devenir en dos problemas. En primer lugar, si se encuentra un cero en el coeficiente $A_{kk}^{(k-1)}$ (i.e: el pivote en el paso k) no se pueden anular los ceros de la columna k, imposibilitando la triangulación. Por ejemplo, sea:

$$A^{(0)} = \begin{bmatrix} 2 & 2 & -1 & 3 \\ -2 & -2 & 0 & 0 \\ 4 & 1 & -2 & 4 \\ -6 & -1 & 2 & -3 \end{bmatrix} \quad b^{(0)} = \begin{bmatrix} 13 \\ -2 \\ 24 \\ -10 \end{bmatrix} \tag{5}$$

Luego del primer paso de la EG la matriz A queda:

$$A^{(1)} = \begin{bmatrix} 2 & 2 & -1 & 3 \\ 0 & 0 & -1 & 3 \\ 0 & -3 & 0 & -2 \\ 0 & 5 & -1 & 6 \end{bmatrix} \tag{6}$$

Dejando el coeficiente $A_{22}^{(1)} = 0$, imposibilitando el siguiente paso. El segundo problema tiene que ver con la elección del pivote. Al realizarse las operaciones descritas con aritmética finita, los coeficientes de la matriz en cada paso no quedan representados de manera exacta. Si se divide a estos coeficientes por un número muy pequeño, el error de representación aumenta considerablemente [4]. Es por ello que para el paso k es conveniente utilizar como pivote aquel coeficiente $m_{jk}^{(k-1)}$ que sea máximo en módulo, pues

de esta manera el cociente de la división (línea 12 del algoritmo 2.1) resulta en un número cercano a cero minimizando el error absoluto al multiplicarse por la fila correspondiente [5].

Basta entonces con intercambiar la fila que contiene al pivote k por aquella que tenga en la misma columna el coeficiente máximo en módulo y luego restar las filas correspondientes para anular la columna debajo del pivote. A este procedimiento se lo denomina **pivoteo parcial**.

2.2. EG con pivoteo parcial

Para su implementación solo basta con agregar que en el paso k-ésimo de la etapa de forward elimination se tiene que evaluar si el coeficiente $A_{kk}^{(k-1)}$ es el de mayor módulo en su columna. Si este nuevo pivote es demasiado pequeño a pesar de ser máximo (i.e: menor a un nuevo parámetro *tolerancia*), se generará una advertencia pues implica que no se puede solucionar el error. Para implementar estas mejoras, luego de la línea 3 del algoritmo 2.1 se realiza la evaluación correspondiente y el intercambio de filas:

Algorithm 3 Modificación de Forward Elimination (agrega pivoteo parcial)

```
1:  for i ← 1 hasta n − 1 do
2:      (...)
3:      filaMax ← i
4:      for s ← i + 1 hasta n do
5:          if |A_{s,i}| > |A_{filaMax,i}| then
6:              maxFila ← s
7:          end if
8:      end for
9:      if maxFila! = i then
10:         A_i ⟷ A_s                ▷ Ahora la el coeficiente A_ii es el máximo de la columna
11:     end if
12:     if A_ii < tolerancia then
13:         ADVERTENCIA: Coeficiente muy bajo, generará error numérico significativo
14:     end if
15:     (...)
16: end for
```

Al igual que la eliminicaión gaussiana sin pivoteo, la complejidad es $\mathcal{O}(n^3)$[7]. Para tratar con el problema de error a profundidad se tiene que tener en cuenta el tipo de la representación numérica que se elige para las operaciones de A. En el problema de difusión las variables $u_i \in \mathbb{R}$, por lo que se debe elegir forzosamente la **representación de punto flotante**.

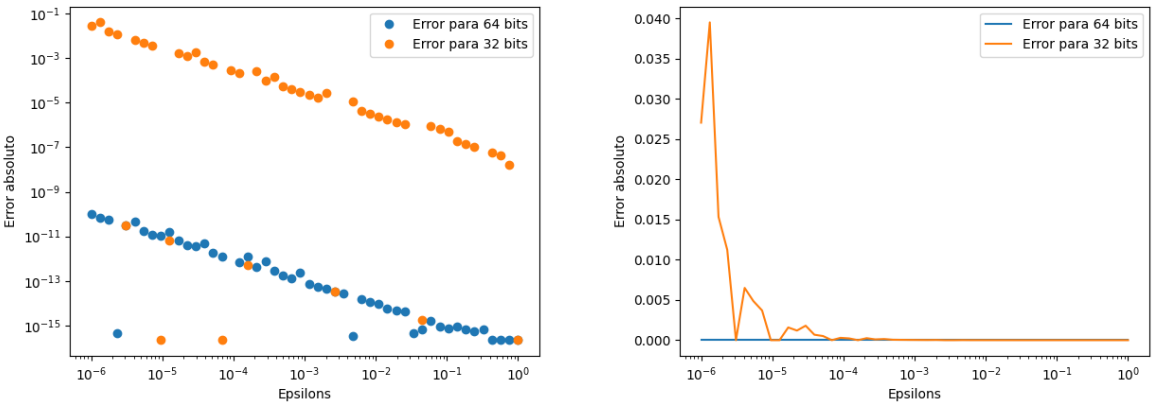
En la representación de números flotantes se aplica el **estándar IEEE** y la resolución y rango de representación está definida por el tamaño en bits del tipo. Por un lado está la representación flotante de **32 bits** y por otro la de **64 bits**. Primeramente es de interés conocer la resolución y rango de números representables de los datos de tipo flotante de 32 y 64 bits. En el caso de 64 bits, la resolución es de 10^{-15} y para 32 bits es de 10^{-6} [6]. Dado que la resolución es $10^{-precision}$, esto quiere decir que la precisión es mucho mayor en 64 bits, lo que implica un error significativamente menor en la conversión. Además, el rango de representación es mucho mayor (en 64 bits el máximo número representable $\approx 1{,}8 \times 10^{308}$ mientras que en 32 bits $\approx 3{,}4 \times 10^{38}$)[6].

Es de esperar, por lo anteriormente expuesto, que la solución calculada $\tilde{x}$ a los sistemas de ecuaciones que estén representados en 64 bits sea mucho más cercana a la solución real x. Si se eligen diferentes casos de $Ax = b$ para los cuales es conocida la solución real x, este fenómeno se puede ver fácilmente tomando la máxima componente de la diferencia entre x y $\tilde{x}$ en módulo. Es decir, tomar la norma infinito $||\tilde{x} - x||_\infty$. Sea $\epsilon \in \mathbb{R}/10^{-6} \leq \epsilon \leq 1$ y el sistema de ecuaciones $Ax = b$ tal que

$$A = \begin{bmatrix} 1 & 2+\epsilon & 3-\epsilon \\ 1-\epsilon & 2 & 3+\epsilon \\ 1+\epsilon & 2-\epsilon & 3 \end{bmatrix} \quad b = \begin{bmatrix} 6 \\ 6 \\ 6 \end{bmatrix} \tag{7}$$

Naturalmente, si $x_i = 1$ se cancelan las constantes y los coeficientes suman 6, dando la solución adecuada. Sin embargo, dado el problema de aritmética finita, la representación de A_{ij} (tal que A_{ij} tenga a ϵ) puede no ser exactamente el valor deseado, provocando que al sumarlo con el otro coeficiente

que contenga a ϵ de la misma fila estos no se cancelen, deviniendo en una solución $\tilde{x}$ distinta de la original.



(a) **(b)**

Figura 1: En ambos gráficos se comparan los ϵ contra $||\tilde{x} - x||_\infty$. En la figura (1a) ambos ejes fueron espaciados de forma logarítmica para facilitar la visualización de los datos. En la figura 1b sin embargo el error se espació linealmente para poner en evidencia la diferencia radical que toma para valores de ϵ muy pequeños.

Como se puede ver en la figura 1 el error decrementa linealmente a medida que ϵ se acerca a 1, pues el coeficiente es más cercano a cero donde hay mayor densidad de números representables [4]. Además, ambas rectas están espaciadas proporcionalmente a la diferencia en la resolución de ambas representaciones. A pesar de que parezca que el error decrece al mismo ritmo, en la imagen 1b se verifica que para valores de ϵ muy cercanos al 0 las operaciones intermedias producen un resultado demasiado alejado al esperado debido a que la conversión ignora a ϵ.

Es por estos motivos que la representación elegida para las matrices será de **punto flotante de 64 bits**, y ya habiendo definido los tipos con los que vamos a trabajar podemos presentar la nueva solución.

2.3. Solución para el Caso Tridiagonal

Como se enseñó previamente, los algoritmos de EG (con o sin pivoteo) tienen un orden de complejidad cúbico, lo que representará grandes restricciones a la hora de evaluar el problema de difusión, pues se requieren M ejecuciones para conocer la evolución del sistema en M pasos. Esto puede ser demasiado costoso para casos grandes de M como es el caso de la mayoría de aplicaciones donde deben considerarse caminatas extensas de partículas.

Dado que la matriz asociada al problema de difusión es tridiagonal, es de esperar que exista un algoritmo que aproveche esta particularidad para anular únicamente la segunda diagonal (la que componen los A_{ij} tales que $2 \leq i < n$ y $j = i + 1$).

A continuación, se expondrá cómo aplicando EG se pueden formular ecuaciones que resumen los cambios de las variables. Sea A una matriz tridiagonal de tamaño n cuya diagonal inferior se representa por el vector a con índices de 2 a n, su diagonal principal se representa por el vector b con índices de 1 a n y su diagonal superior se representa por el vector c con índices de 1 a $n-1$. Sea d el vector que tomará el papel de b en la resolución de $Ax = b$. Se expondrá un ejemplo que ilustre el algoritmo para $n = 4$:

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 \\ 0 & a_3 & b_3 & c_3 \\ 0 & 0 & a_4 & b_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_2 \\ d_4 \end{bmatrix}$$

Se itera en orden por todas las filas i en la matriz ampliada. En el caso $i = 1$, la primera fila se divide por b_i:

$$\begin{bmatrix} 1 & c'_1 & 0 & 0 & d'_1 \\ a_2 & b_2 & c_2 & 0 & d_2 \\ 0 & a_3 & b_3 & c_3 & d_3 \\ 0 & 0 & a_4 & b_4 & d_4 \end{bmatrix}$$

Donde $c'_1 = c_1/b_1$ y $d'_1 = d_1/b_1$ (como estos serán sus valores finales se cambia la notación). Luego, como en la EG, se resta a la fila $i+1$ la i multiplicada por a_{i+1}. En el caso $i = 1$:

$$\begin{bmatrix} 1 & c'_1 & 0 & 0 & d'_1 \\ 0 & b_2 - (c'_1 * a_2) & c_2 & 0 & d_2 - (d'_1 * a_2) \\ 0 & a_3 & b_3 & c_3 & d_3 \\ 0 & 0 & a_4 & b_4 & d_4 \end{bmatrix}$$

Cuando el algoritmo pase a la fila 2, se ejecutarán los mismos 2 pasos y se obtiene $i = 2$. Ahora se divide por b_2, que en este caso ya cambió y es $b_2 - (c'_1 * a_2)$:

$$\begin{bmatrix} 1 & c'_1 & 0 & 0 & d'_1 \\ 0 & 1 & c'_2 & 0 & d'_2 \\ 0 & a_3 & b_3 & c_3 & d_3 \\ 0 & 0 & a_4 & b_4 & d_4 \end{bmatrix}$$

Donde $c'_2 = c_2/(b_2 - (c'_1 * a_2)$ y $d'_2 = (d_2 - (d'_1 * a_2))/(b_2 - (c'_1 * a_2))$.

Notar como ambos pasos no solo crean una matriz triangular superior transformando al vector a en 0 sino que también convierten al vector b en 1. Los únicos coeficientes restantes en la matriz serán c' y d'. Generalizando:

$$c'_i = \begin{cases} c_i/b_i, & \text{si } i = 1, \\ c_i/(b_i - a_i c'_{i-1}), & \text{si } 1 < i < n-1. \end{cases} \tag{8}$$

$$d'_i = \begin{cases} d_i/b_i, & \text{si } i = 1, \\ (d_i - a_i d'_{i-1})/(b_i - a_i c'_{i-1}), & \text{si } 1 < i < n. \end{cases} \tag{9}$$

Este procedimiento es muy similar a la eliminación gaussiana y solo se diferencia en la elección de los coeficientes por los que se multiplica a las filas para simplificar las operaciones. Teniendo esto en mente podemos crear un algoritmo que primero calcule estos coeficientes de 1 a n y luego realice la sustitución hacia atrás como en el algoritmo 2.1.

Algorithm 4 Solución Tridiagonal

```
1:  procedure FORWARD ELIMINATION(a,b,c,d)
2:      n ← dimensión de b
3:      for i ← 1 hasta n do
4:          if i == 1 then
5:              c'_i ← c_i/b_i
6:              d'_i ← d_i/b_i
7:          else
8:              d'_i ← (d_i − a_i * d'_{i−1})/(b_i − a_i * c'_{i−1})
9:          end if
10:         if 2 ≤ i ≤ n − 1 then
11:             c'_i ← c_i/(b_i − a_i * c_{i−1})
12:         end if
13:     end for                ▷ O(n) porque es el único ciclo, dentro hay operaciones de orden constante
14: end procedure
```

El algoritmo redujo el tiempo de cómputo de $\mathcal{O}(n^3)$ a $\mathcal{O}(n)$. Notar cómo todos los coeficientes de c' dependen únicamente de a, b y c, no de d. Los denominadores de d'_i $(b_i - a_i * c'_{i-1})$ tampoco dependen de d. Esto significa que gran parte de los cálculos que lleva adelante el algoritmo no dependen del vector de términos independientes. Precomputando estos valores permite reutilizarlos para resolver los sistemas de ecuaciones que compartan a, b y c, o mejor dicho, la matriz tridiagonal. Esta idea es similar a la de factorización LU.

Algorithm 5 Solución Tridiagonal

```
1:  procedure PRECÓMPUTO(a,b,c)
2:      n ← dimensión de b
3:      C ← arreglo de dimension n − 1
4:      den ← arreglo de dimension n
5:      den_0 ← b_0
6:      C_0 ← c_0/b_0
7:      for i ← 1 hasta n do
8:          den_i ← b_i − (a_i × C_{i−1})
9:          if i != n − 1 then
10:             C_i ← c_i/den_i)
11:         end if
12:     end for
13:     retornar den, C
14: end procedure
```

Así se obtuvo C el precómputo de c' y den el precomputo de $bi - ai * c'_{i-1}$. De esta manera solo basta aplicar las operaciones que se realizan a d [lineas 6 y 8 del algoritmo 2.3] cada vez que este cambie, sin tener que calcular nuevamente las modificaciones a la matriz A. La complejidad asintótica se mantiene, mas es evidente que se está ahorrando tiempo de cómputo porque se realizan las operaciones solo para el vector d. Recordemos que para la simulación utilizaremos constantemente una matriz similar a la matriz tridiagonal del operador laplaciano (que se mantiene constante, razón por la cual podremos aplicar precómputo) y resolvemos el sistema con múltiples vectores $d = u^k$.

2.4. Verificación de la implementación

Para verificar la correctitud de este algoritmo, basta con analizar cómo la matriz laplaciana sirve para resolver sistemas de ecuaciones diferenciales.

El operador de Laplace o Laplaciano es un operador diferencial dado por la divergencia del gradiente de una función escalar. En el caso unidimensional es equivalente a la derivada segunda por lo que se puede usar para resolver ecuaciones diferenciales de la forma $d^2/dx^2 * u = d$, donde la derivada segunda es la matriz tridiagonal del operador laplaciano y d es un vector que va a tomar valores según distintos casos. La forma que tengan las funciones determinará si son respuesta de la ecuación diferencial. Sea $n = 101$ tal que:

$$d_i^{(a)} \quad \begin{cases} 0, \text{ si } i \neq [n/2] + 1 \\ 4/n, \text{ si } i = [n/2] + 1. \end{cases} \tag{10}$$

$$d_i^{(b)} = 4/(n^2) \tag{11}$$

$$d_i^{(c)} = (-1 + 2i/(n-1))12/n^2 \tag{12}$$

Aplicando el algoritmo con estos vectores obtenemos el siguiente gráfico:

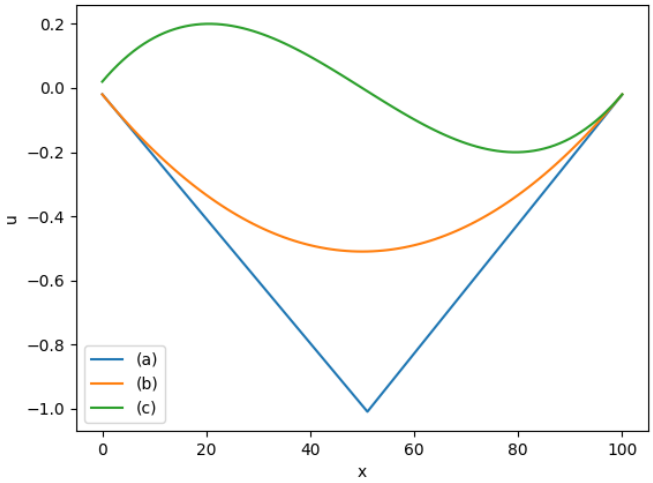


Figura 2: Gráfico de $u^{(1)}$ vs ciertos valores iniciales $x = u^{(0)}$ para cada caso descrito.

El caso (a) es el más particular. Sus valores de d_i son todos 0 a excepción de $i = 51$, en el cual su valor es $4/101$. El hecho de que esta sea siempre 0 a excepción de su punto medio habla de un cambio abrupto que coincide con la forma en V del gráfico. Que el valor d_i sea positivo indica que el cambio es positivo, en términos del gráfico que la función se dispara hacia arriba. Para el siguiente caso los valores de d positivos se traducen en una parábola como la que obtuvimos, pues el hecho de que sean positivos indican la concavidad de la función va a ser hacia arriba y el hecho de que sean constantes indican que la tasa de cambio es uniforme.

Veamos cómo la expresión $(-1 + 2i/(n-1))12/n^2$ refleja correctamente el comportamiento esperado para las funciones cúbicas. En nuestro caso con $n = 101$ tenemos que la tasa de cambio será negativa hasta el valor $i = 50$ donde será exactamente 0. De ahí en adelante los valores son positivos, lo que indica que de la mitad hacia atrás la concavidad será hacia abajo y de la mitad hacia adelante será hacia arriba.

Una vez analizado el comportamiento que deberían tener las funciones u y ya comprobado que se reflejan en el gráfico creado a partir de la nueva implementación, se expondrá a continuación la diferencia de eficiencia de la misma comparado a la eliminación gaussiana con pivoteo parcial.

2.5. Tiempos de cómputo

En virtud de lo anteriormente mencionado se ejecutará el primer paso de la difusión, es decir, calcular el vector $u^{(1)}$ a partir de $u^{(0)} = 0$ para distintos tamaños de u y se tomará el tiempo que tardó en encontrar la solución. Como la EG gaussiana es de orden cúbico y el algoritmo específico para resolución de sistemas con matrices tridiagonales es lineal, una buena manera de exponer la diferencia sería comparando los tiempos de ejecución para diferentes tamaños de matrices y mostrar los datos en escala logarítmica. Esto debido a que quedan ambos algoritmos plasmados en rectas que facilitan la lectura, y solo con ver la diferencia en la pendiente se puede discernir a simple vista la mejoría.

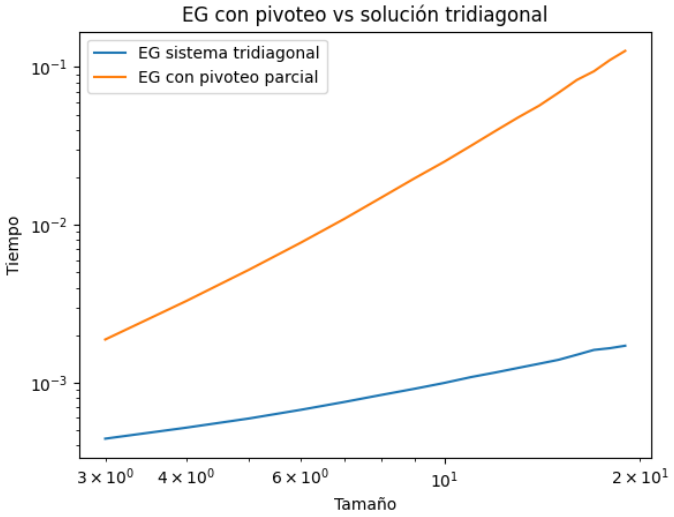


Figura 3: Se expone el tamaño de u contra el tiempo absoluto que tardó cada algoritmo. Ambos ejes están espaciados en escala logarítmica, razón por la cual se observa que ambas funciones aproximan lineas rectas.

Queda evidenciado el cambio de pendiente en el caso del algoritmo con pivoteo, pues es una función de orden cúbico, a comparación del tridiagonal donde tiempo de ejecución aumenta proporcionalmente al tamaño de la entrada. Sin embargo, a pesar de que el orden de complejidad es inmejorable (pues es lineal) como se mencionó anteriormente existe una variante que realiza un pre-cómputo de los coeficientes para facilitar la ejecución del algoritmo. Esto mantiene la linealidad del mismo en términos de complejidad asintótica, pero decrementa notablemente la cantidad de operaciones que se realizan como fue demostrado.

Para probar lo anteriormente mencionado, a continuación se expondrán los tiempos que toma ejecutar n veces el algoritmo con matrices de dimensión n para cada caso.

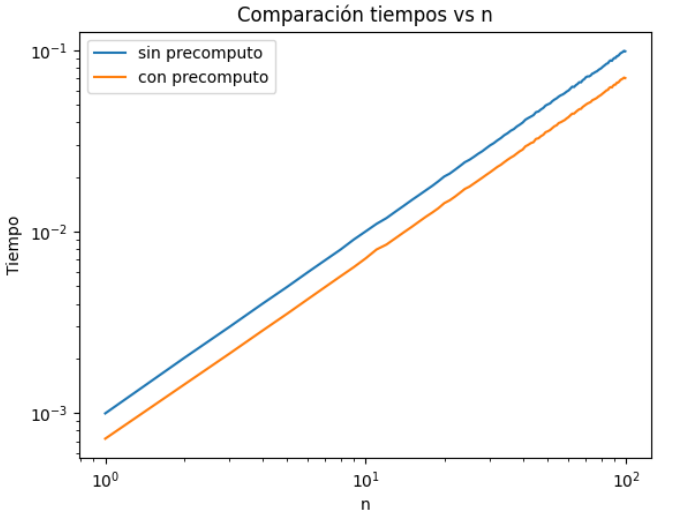


Figura 4: Se expone la dimensión de la matriz vs el tiempo que tomó ejecutarse n veces la operación $Au_1 = 0$.

La pendiente es la misma porque el orden de complejidad de las implementaciones sigue siendo lineal. Sin embargo, las rectas están separadas mostrando que la implementación que usa el precómputo es más rápida. Sumado a esto, el hecho de que los ejes sean logarítmicos implica que esta diferencia es más notable a medida que aumenta n, por lo que para dimensiones grandes de matrices es una mejora radical.

Con esta mejora ya podemos resolver nuestro problema de manera eficiente. Para probarlo a continuación llevaremos a cabo múltiples simulaciones de difusión basándonos en las ecuaciones (2) y (3).

2.6. Simulación de difusión

Recordemos que para expresar la ecuación de difusión consideramos el incremento para el paso k, $u^k - u^{k-1}$ como una fracción del operador la laplaciano aplicado a u^{k-1}:

$$u_i^{(k)} - u_i^{(k-1)} = \alpha(u_{i-1}^{(k-1)} - 2u_i^{(k-1)} + u_{i+1}^{(k-1)}) \tag{13}$$

$$u_i^{(k)} - u_i^{(k-1)} = \alpha u_{i-1}^{(k-1)} - 2\alpha u_i^{(k-1)} + \alpha u_{i+1}^{(k-1)} \tag{14}$$

$$u_i^{(k)} = \alpha u_{i-1}^{(k-1)} + (-2\alpha + 1)u_i^{(k-1)} + \alpha u_{i+1}^{(k-1)} \tag{15}$$

Que puede representarse como el sistema:

$$\begin{bmatrix} (-2\alpha+1) & \alpha & 0 & 0 \\ \alpha & (-2\alpha+1) & \alpha & 0 \\ 0 & \alpha & (-2\alpha+1) & \alpha \\ 0 & 0 & \alpha & (-2\alpha+1) \end{bmatrix} \begin{bmatrix} u_1^{k-1} \\ u_2^{k-1} \\ u_3^{k-1} \\ u_4^{k-1} \end{bmatrix} = \begin{bmatrix} u_1^k \\ u_2^k \\ u_3^k \\ u_4^k \end{bmatrix} \tag{16}$$

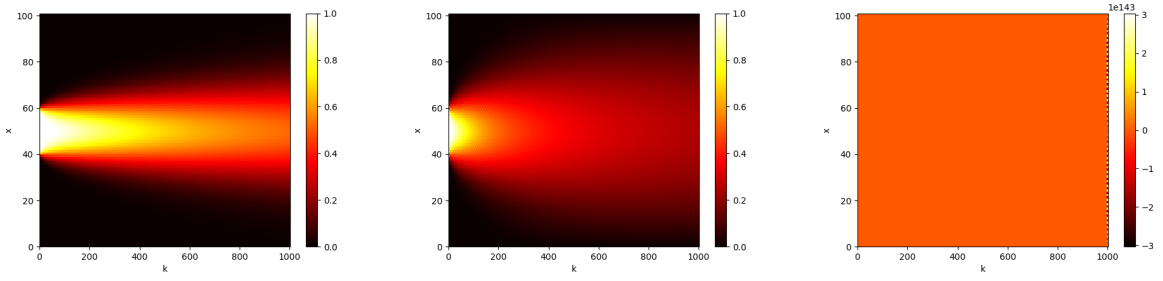
Es decir, contando con un estado u^i podemos obtener u^{i+1} con un producto matricial.:

$$u^k = Au^{k-1} \tag{17}$$

Aplicando esta ecuación se pueden ejecutar simulaciones que muestren la evolución en densidad de un sistema de partículas con cierto coeficiente de dispersión α. Sea n la cantidad de posiciones tal que $n = 100$ y $m = 1000$ la cantidad de pasos. Definimos $r = 10$ como una constante para definir al área inicial de las partículas tal que:

$$u_i^0 = \begin{cases} 0, \\ 1, \quad \text{si } [n/2] - r < i < [n/2] + r. \end{cases} \tag{18}$$

Con estas condiciones veamos cómo varía el gráfico de la simulación para ciertos valores de α:



(a) **(b)** **(c)**

Figura 5: Los graficos modelan las densidad de partículas en la posición x en cada paso k. En la figura 5a $\alpha = 0,1$, en la 5b $\alpha = 0,5$ y en la 5c $\alpha = 0,6$.

A simple vista se ve el efecto deseado, que es que la difusión es proporcional a α. Mientras que en el primer gráfico las partículas se mantienen cercanas al centro (i.e: a la posición inicial), en el segundo gráfico la concentración de partículas en el centro disminuye a medida avanzan.

Esta formulación explícita no genera soluciones estables para todos los valores de alpha. Esto se debe a la acumulación de error numérico provocando que la magnitudes crezcan de una manera desproporcional a lo que realmente deberían, provocando que con ciertos valores (por ejemplo, $\alpha = 1$) no se pueda ejecutar el algoritmo porque los x se van a infinito. Analizando más a detalle con valores que funcionan, se espera que con un coeficiente de dispersión α levemente superior al caso 5b la misma aumente proporcionalmente. Sin embargo, en la figura 5c se ve cómo las partículas parecen tener una concentración uniforme a lo largo del espacio mostrando así el error de representación.

Analicemos entonces el **caso implícito** que se deriva de la formulación que considera que el incremento depende de la laplaciano aplicado en la magnitud en el paso k en vez de $k-1$.

Para obtener la forma matricial se aplica un procedimiento símil al caso explícito:

$$u_i^{(k)} - u_i^{(k-1)} = \alpha(u_{i-1}^{(k)} - 2u_i^{(k)} + u_{i+1}^{(k)}) \tag{19}$$

$$u_i^{(k-1)} = -\alpha u_{i-1}^{(k)} + (2\alpha+1)u_i^{(k)} - \alpha u_{i+1}^{(k)} \tag{20}$$

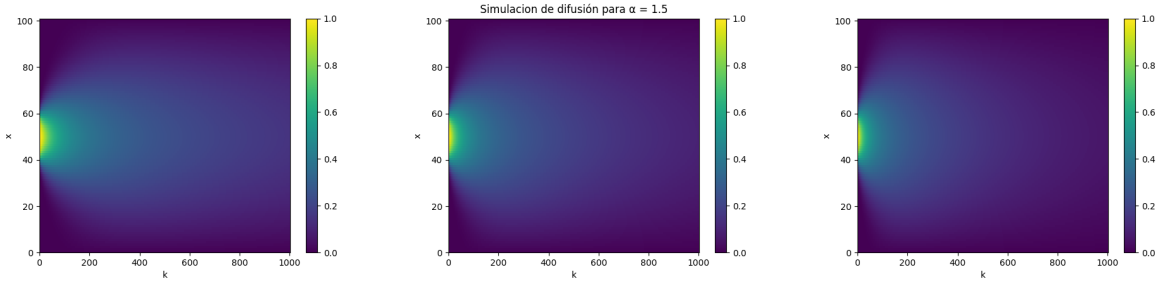
Que se representa con el sistema:

$$\begin{bmatrix} (2\alpha+1) & -\alpha & 0 & 0 \\ -\alpha & (2\alpha+1) & -\alpha & 0 \\ 0 & -\alpha & (2\alpha+1) & -\alpha \\ 0 & 0 & -\alpha & (2\alpha+1) \end{bmatrix} \begin{bmatrix} u_1^k \\ u_2^k \\ u_3^k \\ u_4^k \end{bmatrix} = \begin{bmatrix} u_1^{k-1} \\ u_2^{k-1} \\ u_3^{k-1} \\ u_4^{k-1} \end{bmatrix} \tag{21}$$

Es decir:

$$u^{k-1} = Au^k \tag{22}$$

Como u^k es una incógnita y conocemos el estado inical u^0, para ver la densidad en cada posición en el paso siguiente ya no basta con un producto sino que hay que resolver el sistema.



(a) **(b)** **(c)**

Figura 6: Evolución de densidad en 1000 pasos para $\alpha = 1$ (figura 6a), $\alpha = 1,5$ (figura 6b) y $\alpha = 2$ (figura 6c)

La representación implícita es la más adecuada para modelar el problema pues se puede apreciar que representa correctamente el aumento proporcional de la dispersión respecto al coeficiente α. Como consecuencia, es la única manera viable de traducir el estudio de la difusión promedio de partículas a la resolución de sistemas de ecuaciones. Otro motivo para descartar la forma explícita es que si bien parece más simple porque consiste en realizar productos matriciales, estos tienen un costo cuadrático en comparación al costo lineal que ofrece la alternativa. Por lo tanto incluso para los rangos de α en donde la forma explícita no falla ($0 \leq \alpha \leq 0,5$) conviene utilizar su alternativa implícita.

3. Conclusiones

Se expusieron diversos algoritmos para resolver el problema de la difusión de partículas y el estudio de la evolución del sistema midiendo la densidad en cada paso. Al modelarlo como un sistema de ecuaciones y expresarlo con la matriz asociada correspondiente a la formulación explícita e implícita se obtuvieron distintos resultados.

Se evidenció la importancia de manejar adecuadamente el error numérico que es inevitable en la aritmética finita. Si se usa el tipo de datos correcto se pueden representar todos los valores necesarios y a su vez conservar precisión para dar soluciones en un rango aceptable respecto a las verdaderas. En este caso, la conversión a punto flotante de 64 bits fue suficiente para modelar el problema de difusión y se vieron las limitaciones que imponen otros tipos como fue el caso de 32 bits, que despreciaba cambios leves. Al estar computando varias operaciones concatenadas (i.e: que dependen del resultado de la anterior), el hecho de ignorar pequeños valores puede devenir en soluciones radicalmente distintas como se evidenció en la figura 1b

Relacionado al problema de difusión en sí, algo importante a destacar es que la discretización del mismo y su consecuente representación como una matriz tridiagonal permitieron el desarrollo de una alternativa que resuelve el sistema en un orden de complejidad significativamente menor, pudiendo así realizar las simulaciones correspondientes para obtener la evolución del sistema de una manera eficiente. Esto es de vital importancia, pues la naturaleza misma del problema implica la ejecución del algoritmo reiteradas veces. Como este problema interpela a diversas áreas en las que se trabajan con muchos datos esta diferencia ahorra mucho tiempo, facilitando así su investigación.

Otro detalle no menor a destacar es que en el ámbito de la computación muchas veces se decide sacrificar precisión en las soluciones a cambio mejorar la complejidad del algoritmo que las calcula, lo que no sucede en este caso. El error de la solución computada va a estar definido solo por el error que producen las operaciones por estar trabajando con aritmética finita, mas no está vinculado de ninguna manera con el cambio de la solución en sí.

Cualquier problema que sea pueda modelar como un sistema de ecuaciones que tiene a una matriz tridiagonal asociada puede ser resuelto en tiempo lineal con este algoritmo. Sin embargo, para cualquier otra matriz no se puede aplicar una solución de complejidad asintótica menor a la cúbica (o cuadrática si se implementa LU).

4. Bibliografía

Referencias

[1] Artículo de la máquina de Galton

[2] Langtangen, H. P. (2013). Finite difference methods for diffusion processes. University of Oslo

[3] Dato del enunciado del trabajo práctico

[4] Primera clase del laboratorio, error numerico

[5] Análisis numérico, Richard L. Burden, J. Douglas Faires y Annette M. Burden [Capítulo 6.2]

[6] Informacion de tipos de Numpy

[7] Clases teóricas

5. Apéndice

Para la ejecución de las simulaciones y gráficos que se expusieron se usó el lenguaje **Python** con la biblioteca **Numpy** para los cálculos matriciales. Todos los gráficos expuestos en el informe fueron elaborados con la biblioteca **matplotlib** y el argumento de tolerancia del algoritmo de eliminación gaussiana con pivoteo es 10^{-6}.

Para las mediciones de tiempo se usó la biblioteca **timeit** y con el fin de asegurar la veracidad de los tiempos de ejecución de los algoritmos expuestos para cada caso se tomaron cinco medidas y se eligió la que tardó menos. Cada medida consiste en el promedio de cien ejecuciones. A partir de cinco mediciones los resultados comienzan a converger, razón por la cual es una cantidad de muestras suficiente para los problemas por los que se realizaron las medidas.